09/743520

JC07 Rec'd PCT/PTO 11 JAN 2001

DATA PROCESSING APPARATUS ~~AND METHOD FOR~~ OPTIMISING CONFIGURATION PARAMETERS OF A PHYSICAL SYSTEM

The present invention generally relates to the data processing method and apparatus for determining optimum parameters of the model of a physical system and for controlling the configuration of the physical system.

The generation of a model of a physical system is used for a wide variety of applications including learning more about the behaviour of the system and controlling the parameters of the system. The model of the physical system can be used to try out parameters in order to determine optimum parameters which can then be used for controlling the physical system. In this way the effect of the parameters on the physical system can be predicted so that only optimum parameters are chosen for use in the control of the physical system.

The theory and motivation behind optimisation of systems has been considered by many workers, notably David E Goldberg. In "Genetic Algorithms in search, optimisation and machine learning", Goldberg presents the goals of optimisation as an "improvement of system performance towards some optimal point or points". Goldberg describes several approaches that can be applied to achieve improvements in performance and presents methods for measuring system performance. The approaches presented include representation of a problem - coding of parameter sets defining the problem - and the application of genetic operators such as mutation and cross over to the coded parameter sets.

One such physical system for which configuration parameters are required is a distributed database. Because the use of corporate intranets continues to rise, the management of efficient access to data is becoming a key issue. Large information systems, often accessed on a global basis, are increasingly being provided by distributed means with use of "mirror sites" to ease congestion and improve local access. The configuration of the information system defining the location of the

information and the servers to which clients are to direct queries is important in order to provide a system which has a high performance as perceived by the clients. Further, where access is truly global, static design solutions are not ideal, as the largest source of load on the system moves from geographic area to geographic area at different times of the day. This tends to create congestion that is "localised" in the region of demand. Localised mirroring in one region only will simply shift the contention onto the global communications networks when demand shifts to other regions. Mirroring in all regions is not only costly, but also leads to over duplication of data, and greater problems with integrity and the need to perform multiple updates of the data.

Where the data is required to be both updated as well as read, the administration of these information systems can become significant and labour intensive. In a recent paper by MJ Oates *et al* entitled "Investigating Evolutionary Approaches for Self-Adaption in Large Distributed Databases" (Proceedings of the 1998 IEEE International Conference on Evolutionary Computation) and in a paper by G Bilchev *et al* entitled "Comparing Evolutionary Algorithms and Greedy Heuristics for Adaption Problems" (Proceedings of the 1998 IEEE International Conference on Evolutionary Computation) it has been shown that autonomous management of distributed information systems is feasible and in these publications

the use of various types of algorithms for the adaption of distributed databases has been considered.

The present invention provides an improved method of apparatus for determining optimum parameters of the model of a physical system wherein at least one initial string of values representing the parameters of the model to be optimised is obtained; a cost value associated with the model having parameters represented by the reached string of values is determined; a new string of values is repeatedly generated by a) selecting a sequence of values of random length starting at a random position in a string of values, b) replacing the sequence of values of the same length in a string of values at a random position and c) changing the value of one more of the values of the resulting string of values to generate a new string of values; a cost value associated with the model having parameters represented by the new sting of values is determined; and the optimum parameters are determined as one of the initial or new string of values for which the cost value associated with the model having the optimum parameters is closest to a target, such as a maximum or a minimum.

Using this technique new strings of values representing new parameters of a model are produced which can be applied to the model to see if a cost value is closer to a target e.g. a minimum or a maximum is

produced. The technique thus provides a novel search technique to search for the optimum parameters.

This technique is particularly applicable for the control of the configuration of a physical system wherein the model models the physical system and the new parameters generated represent new untried configurations of the physical system. The cost value output from the model can be used to determine whether the new configurations are an improvement or not. Once the optimum parameters have been determined they can be used to control the configuration of the physical system.

The present invention is particularly suited to the problem of configuring a distributed database comprising a plurality of data servers connected over a network, each data server holding any number of data units, and a plurality of clients connected over the network to the data servers, each of said clients being adapted to retrieve data units from and/or update data units at one or more of the data servers. When the invention is applied to the configuration of a distributed database, the configuration parameters define which data server is to be accessed by which client and the distributed database is modelled using the configuration parameters and information on the passage of data within the distributed database to determine a performance value for each of the strings of values. Which ever of the strings of values have the best performance value is then used

for configuring the distributed database to control which data server is to be accessed by which client. The performance value is representative of the performance perceived by one or more of the clients. In this aspect of the present invention a client comprises any application running on a computer connected to the network which requires access to the data units stored on the data servers. Where the performance value represents the transaction time for one or more clients accessing a data unit on a data server, the best performance value is a minimum and thus the technique of this embodiment of the present invention will attempt to minimise the performance value i.e. the transaction time.

Although the present invention is particularly applicable to optimising the configuration of a distributed database, it can be applied to any physical system or model wherein parameters of the model can be provided as a string of values. The values can either comprise actual numerical values of the model or can comprise numerical values representing features of the model i.e. they comprise or represent symbols.

The present invention can be applied generally to the configuration of a distributed processing system, of which a distributed data base is but one example, wherein the configuration parameters determine which nodes (servers) in the network are to be used for processing requested by applications (clients).

In a preferred embodiment of the present invention the string of values is considered to wrap round to form a continuous string such that the last and first values of the string are sequential. In this way, when a sequence of values at random length is selected in a string of values the sequence can include the last and first values sequentially. Similarly, when a sequence of values is replaced in a string of values this can include the sequential last and first values.

By selecting a sequence of values and overlaying it the intention is that good segments of the string of values in one part of the sting may also work well as a building block elsewhere in the string. This technique allows the values which may have become "extinct" in positions in the population of strings to be "reinjected" thereby enhancing the searching technique. The technique for generating the initial parent strings of values does not appear to be important. In one embodiment these are generated randomly.

The present invention is applicable to evolutionary computational search techniques and other conventional search techniques. For example, the technique of the present invention can be applied to a genetic algorithm wherein a population of strings of values is created and new strings of values are created from two parents. The technique of the present invention when applied to genetic algorithms is a cross-over technique wherein a

sequence of values of random length is selected at a random start position in a first string of values and this is used to replace a sequence of values at the same length in a second string of values at a random position. One or more of the values of the resulting string of values is then mutated to generate the new string of values. This basic technique can be used in any sort of genetic algorithm such as a Breeder genetic algorithm or a Tournament genetic algorithm.

As mentioned above, the present invention is applicable to non-evolutionary search techniques in which a single string of values is used as a parent from which a new string of values is generated. In this method the sequence of values of random length is selected starting at a random position in the parent string of values and this is used to replace a sequence of values in the same length in the parent string of values at a random position. One or more of the values of the resulting string of values is then changed to generate the new string of values. This basic technique can be used in a Hill Climbing search technique or in a Simulated Annealing search technique for example.

Since the strings of values comprise a number of values, each of which is a value for a particular parameter, the strings of values can be termed solution vectors since they can be represented as n dimensional

vectors and the "best" one will comprise the solution to the problem of finding the best configuration parameters.

Embodiments to the present invention will now be described with reference to the accompanying drawings, in which:

Figure 1 is a schematic illustration of the use of the technique of the present invention to determine optimum parameters of a model in the physical system,

Figure 2 is a schematic diagram of use of the technique of Figure 1 to control the configuration of a physical system,

Figure 3 is a schematic diagram illustrating the use of the technique to control the configuration of a distributed database in accordance with one embodiment of the present invention,

Figure 4 is a schematic diagram of a distributed database in accordance with one embodiment of the present invention,

Figure 5 is a schematic diagram of a distributed database in accordance with another embodiment of the present invention,

Figure 6 is a schematic diagram of a distributed database in accordance with a further embodiment of the present invention,

Figure 7a is a flow diagram illustrating a retrieval operation,

Figure 7b is a flow diagram illustrating an update operation,

Figure 8 is a flow diagram illustrating steps in adapting the distributed database,

Figure 9 is a schematic diagram of a distributed database in accordance with a first scenario,

Figure 10 is a schematic illustration of a solution vector,

Figure 11 is a flow diagram of the steps in a first "basic" method for using a model to determine a cost value,

Figure 12 is a flow diagram of a Breeder genetic algorithm for determining the lowest cost value,

Figure 13 is a flow diagram of a single three way Tournament genetic algorithm for determining the lowest cost value,

Figure 14 is a flow diagram of the steps for generating a new solution vector in accordance with an embodiment of the present invention,

Figure 15 is a schematic illustration of the process for generating a new solution vector in accordance with the flow diagram of Figure 14,

Figure 16 is a schematic illustration of a distributed database of a second scenario (B),

Figure 17 is a flow diagram illustrating the steps of a second method termed "just used" for using the model to determine a cost value,

Figure 18 is a flow diagram illustrating the steps of a third method termed "plus average" for using the model to determine a cost value,

Figure 19 is a flow diagram illustrating the steps of a fourth method termed "plus used" for using the model to determine a cost value,

Figure 20 is a flow diagram illustrating the steps of a fifth method termed "just used" for using the model to determine a cost value,

Figure 21 is a flow diagram illustrating the steps of a sixth method termed "plus all" for using the model to determine a cost value,

Figure 22 is a flow diagram illustrating the steps of a seventh method termed "plus 10%" for using the model to determine a cost value,

Figure 23 is a flow diagram illustrating a Hill Climbing algorithm,

Figure 24 is a flow diagram illustrating a Simulated Annealing algorithm,

Figure 25 is a flow diagram illustrating the steps of one embodiment of the present invention for generating a new solution vector,

Figure 26 is an illustration of the technique of Figure 25 for generating a new solution vector,

Figure 27 is an illustration of the results for five different optimisation techniques for scenario A using the basic, least worst server model,

Figure 28 is an illustration of the results for the five different optimisation techniques for scenario A using the "plus all" model,

Figure 29 is an illustration of the results for the five different techniques for scenario B using the basic, least worst model, and

Figure 30 is an illustration of the results for the five different techniques for scenario B using the "plus all" model.

Figure 1 is a schematic illustration of the principles of one aspect of the present invention. A modeller 50 operates to generate a model of a physical system. In order to generate the model the modeller 50 receives configuration parameters from an optimiser 60. Thus, the optimiser 60 controls the configuration of the model generated by the modeller 50. As a result of the generated model, the modeller 50 outputs a cost value to the optimiser 60. The cost value can represent any function of the model which it is desired to, for example, reduce or increase. For example, in the design of a system it may be desirable for the cost value to represent the energy consumption of the system which is to be reduced or the performance of the system which is to be improved (increased). The optimiser 60 operates a form of a search algorithm in order to generate a string of values representing the configuration

parameters. The string of values is termed a solution vector since it can be considered to be an n-dimensional vector having values representing the configuration parameters. The optimiser 60 adaptively changes the configuration parameters in order to optimise the cost value i.e. either reduce it or increase it. The search for a solution vector which will provide the optimum cost value involves the copying of a segment of a solution vector of random length and starting at a random position as an overlay for a segment of the solution vector of the same length at another random position. One or more of the values of the solution vector is then chosen randomly and changed. This process generates a new solution vector representing a new set of configuration parameters used by the modeller 50. The generation of the new configuration parameters using this technique provides an improved method of searching for the optimum cost value.

Figure 2 is a schematic illustration of a second aspect of the present invention wherein the technique of Figure 1 is applied to the control of the configuration of a physical system 70. The modeller 50 and optimiser 60 operate as described hereinabove with reference to Figure 1. Once the optimiser has determined an optimum cost value, the configuration parameters corresponding to the optimum cost value are output to the physical system 70 in order that the system be optimumly

configured. In order for the modeller 50 to accurately model the physical system 70, the physical system outputs operating parameters to the modeller 50. In this aspect of the present invention the use of the modeller 50 enables the prediction of the outcome of changes in the configuration of the physical system before the changes are actually made to the physical system 70. In other words, no non-optimum changes need be made to the configuration of the physical system 70 in order to try to achieve an optimum configuration. The determination of the optimum configuration is predicted using the model.

The operating parameters output from the physical system 70 to the modeller 50 can comprise any measured data which is required for the accurate modelling of the physical system by the modeller 50. Such data can be measured from the physical system.

Figure 3 is a schematic illustration of an aspect of the present invention wherein a distributed database is optimumly configured. This system is as described with reference to Figure 2 except that the physical system is a distributed database comprising data servers 95 and terminals 90 operating clients requiring access to data on the databases 95 and interconnected via a network 80. The modeller 50 receives usage data indicating the pattern of usage of the data in the databases 95 by the client's in the terminals 90. The

modeller 50 also requires information on communication speeds in order to accurately model the distributed database. The optimiser 60 will search for an optimum solution vector representing configuration parameters for the distributed database which indicate the optimum usage pattern i.e. which databases 95 the clients on the terminals 90 should look to for retrieval of data. if their usage patterns perform a repeated cycle or are predictable, projected usage data can be fed into the model instead of retrieved from the network 80 thereby allowing the system to generate the new configuration parameters based on anticipated work load. The data units held in the databases 95 can comprise any form of information such as text, audio, video or images, or any other form of database information.

The distributed database can be provided over a local area network such as an ethernet or a corporate intranet, or over a wide area network such as the internet.

The distributed database can either be a homogenous database wherein all of the data units are of the same format, or it can comprise a heterogenous database wherein data units are stored in different formats. In a heterogenous database it will be necessary to translate between the different formats of the data units in order for data to be passed between databases e.g. during data "mirroring" operations, and during retrieval and update

operations if the format of the data operated on by the client is different to the format of data in the database.

The reconfiguration of the database based on the configuration parameters results in a client being directed to a particular data server for data. When the configuration is changed the client is directed to a new data server, if the data server does not contain the required data it is necessary for the data to be copied from a data server containing that data. Thus, the reconfiguration of the distributed database comprises a mix of simply redirection of queries from a client to a database server and the copying of data between data servers. Since the copying of data between data servers requires a high band width over the network, this operation is minimised where possible. One scenario, where the copying of data between data servers takes place is when a data server becomes heavily used by clients. The "mirroring" of the data to another data server will spread the load between the data servers thereby enhancing the performance of the distributed database.

The adaption of the configuration of the distributed database can either take place continuously during the operation of the distributed database, at a period when the use of the distributed databases are low, or at predetermined times.

The modeller 50 and optimiser 60 are preferably implemented as a computer program operating on a computer. In order to reduce the amount of processing that is carried out, the calculation of the optimum configuration parameters can be triggered to occur only in response to a suitable criteria such as the failure of a data server, a network link failure or change, or when the time taken by one or more of the clients to retrieve and/or update data units at one or more of the data servers exceeds a threshold. This avoids the adaption process continuously hunting for an optimum solution when the distributed database may already be optimumly configured or near optimumly configured. Alternatively, the adaption algorithm can operate continuously but the configuration determined may only be applied if the predicted improvement in performance exceeds a threshold. Since the overheads associated with reconfiguration can be significant, a balance must be struck between accepting sub-optimal performance and the frequency of reconfiguration.

Figure 4 illustrates a distributed database in one embodiment of the present invention wherein the first database 1 (DB1) and the second database 2 (DB2) are connected to a network 3 and thus form nodes in the network. The first database 1 is a relational database and includes a relational database management system (RDBMS) 4. The second database 2 is an object oriented

database and includes an object oriented database management system (OODBMS) 5. The database management systems 4 and 5 manage the respective database e.g. control access, security, backups etc. and operate to interface the databases 1 and 2 to the network 3.

A first network station 6 is connected to the network 3 and operates a first application APP1 which is interfaced to the network 3 by an access manager AM1. A second work station 7 is also connected to the network 3 and runs a second application APP2 which is interfaced to the network 3 via an application manager AM2. The applications APP1 and APP2 include performance files 8 and 9 for the storage of the monitored performance during retrieval and/or updating of data to the databases 1 and 2. The access managers AM1 and AM2 include respective usage files 10 and 11 for storing information on the usage of databases 1 and 2. Also the access managers AM1 and AM2 include translators 12 and 13 for translating queries and data between database formats when necessary.

The database network also includes a controller 14 which operates an adaptor application 15 for carrying out the adaption algorithm for the control of the configuration of the database. The adaptor application 15 is interfaced with the network 3 via an adaptor application access manager AM3. The access manager AM3 includes a translator 16 for translating data between different database formats where necessary and a parser

for parsing generic messages to move, copy and/or delete data. The controller 14 also contains tables 17 which contain data identifying the location of data files in the databases, information identifying the type of format of the database and the location, and information identifying the database containing information which may be accessed by each application APP1 and APP2.

Although in Figure 4 only two work stations 6 and 7 and two databases 1 and 2 are illustrated, any number can be provided over the network 3. Also only work stations 5 and 6 are illustrated as only operating single respective applications APP1 and APP2, they can, in a multitask environment, be operating more than one application which can retrieve and/or update data i.e. the work stations 5 and 6 can host more than one client. The respective access manager AM1 and AM2 identifies which of the applications in a respective work station 6 and 7 has requested or sent data and it is routed accordingly.

Within Figure 4 the adaptor application 15 and the table 17 are illustrated as being provided in a separate controller arrangement 14. However, these can be distributed throughout the work stations 6 and 7 as is illustrated in Figures 5 and 6.

Figure 5 illustrates an embodiment wherein the work stations 6 and 7 include tables 17a and 17b which are copies of tables 17. This arrangement is advantageous

since the access managers AM1 and AM2 can access the tables directly without incurring a network delay.

Figure 6 illustrates a further embodiment wherein the work stations 6 and 7 also run local adaptor applications 15a and 15b and thus the adaption algorithm is carried out in a distributed manner. Local parsers 20a and 20b are provided in the respective access managers AM1 and AM2 for parsing generic messages from the respective adaptor applications 15a and 15b. This distribution of the adaptor application will benefit simply from parallel processing, or the adaptor application 15a and 15b could operate on a localised basis to control distribution of data locally i.e. only at a few of the databases.

Methods of adaptively controlling the configuration of the distributed databases will now be described.

Figure 7a illustrates the steps carried out when data is requested from a database by an application on a work station. In step S1 the application generates a query for data which includes the data identification (ID). In step S2 the access manager responds to the query and looks in the tables to determine the application location, the database location for the applications and the identity and type of database containing data. In step S3, based on the information from the tables, it is determined whether there is a difference in format between the format of the database and the format of the

application. In step S4 if there is a difference in format the query is translated and in step S5 the data is obtained from the database. In step S6 the access module records the database usage in the usage file. If the format of the database is different to the format of the application, in step S7 it is determined whether the translation of the data is necessary and in step S8 the translation of the data takes place if necessary. In step S9 the data is returned to the application by the access manager. In step S10 the application records, in a performance file, how long the transaction took to complete which is an indication of the performance of the retrieval operation.

In step S11 the adaptor application reads the performance and usage files and in step S12 the adaptor application can determine whether an adaption of the distributed database is necessary. This adaption can only take place off line, continuously in real time, only at predetermined times, or only when network traffic will permit the transference of data over the network between databases.

The usage file used in step S6 contains an identification of the data record retrieved from the database, an identification of the database from which it was retrieved, an identification of the application which retrieved the data record, and the date and time of retrieval. The usage file can either comprise a

single file for databases, or it can comprise a single file for each of the databases. This is used for determining an optimum configuration for the distributed database.

The performance files comprise a file for each application. The performance files contain data identifying the retrieved record data, the application identification, the query identification, and the transaction duration. These are used for determining whether to determine an optimum configuration for the distributed database.

In the embodiments described hereinabove the tables 17, 17a and 17b can comprise a number of different tables containing information used for identifying the location of data in the distributed database. The system configuration table can contain data which identifies the database, the database type e.g. a relation database or an object oriented database, and the connect string used to connect to the database over the network. The advantage of using such a table is that when a new database is added to the distributed database network, its details can simply be added to this file and the algorithm, as will be described hereinafter, can adapt the network to efficiently accommodate it.

The tables can also include a data allocation for each work station which records where an application on a network station should look for each data record. In

these embodiments of the present invention these tables specify where the information can be retrieved for an application.

The tables can also include a data configuration table which lists all of the locations of each type of data record.

The table can further include an application configuration table which is used by the access manager so that it can determine where each application is running and so which data allocation to use. The table thus containing the application identification and the identification of the work station on which it is running.

Figure 7b is a flow diagram of steps carried out for an update. In step S100 the application generates an update query and identifies the data type which is to be updated. The access manager receives the query and data from the application in step S101 and looks in the tables to determine the application location, the database locations containing the data type and the identity of the databases containing the type of data. In step S102 it is determined whether it is necessary to translate the update query and the data from one database to another depending upon the database format of the data generated by the application and the data format of the type database to be updated. If the translation between the database format is required, in step S103 the update

query and data are translated. In step S104 the data is then transferred to the databases containing the data type to update the data therein. In step S105 the access manager records the database usage in the usage file and in step S106 the database management system returns a configuration of the update of the application. The application then records the performance in the performance file in step S107 and in step S108 the adaptor application reads the performance and usage file. If the adaptor application determine that the performance of the distributed database can be improved, in step S109 the adaptor application adapts the distributed database.

Thus in accordance with the update method, all of the database types which corresponds to the type of data generated by the application are updated for subsequent access by any other application in the distributed database.

The adaptor algorithm in accordance with one embodiment of the present invention will now be described.

Applications access data either on a local server or, via a communications link, on a remote server. The best location of data depends upon the concentrations of demand and the relative performance cost of each server and the associated communications costs. An optimum configuration may require data replication so that the overall demand for given data can be shared across

multiple servers and so reduce load and performance costs for a given server. Whilst the distribution of data across multiple servers is desirable and advantageous for data retrieval, this poses difficulties for updating data since data must be updated at all servers. The algorithm aims to compute an improved cost solution to cause a reconfiguration of the distributed database when overall performance figures have exceeded an undesirable threshold. Other factors can trigger the adaption such as the failure or anticipated shut down of one or more communication links or databases, or communication costs for a communication link exceeding a threshold.

The following algorithm, in accordance with one embodiment of the present invention, computes costs into in terms of response times i.e. the time between initiating the request for data and the data being received.

The steps of the adaptation method of one embodiment of the present invention will now be described with reference to Figure 8.

In step S20 the adaption algorithm for the adaptor application is triggered into operation by a threshold decrease in performance being exceeded. In step S21 the adaptor application calculates the optimum distribution of data of the network and the optimum application to data links taking into consideration the model parameters as will be described hereinafter. In step S22 the

adaptor application determines whether there are differences in the optimum distribution data and the current distribution of data and generates messages to copy/move and/or delete data at database nodes in the network and to update links either define the nodes which are to receive queries from the applications i.e. where an application is to receive data. The adaptor application access manager receives the messages in step S23 and parses each message. In step S24 the adaptor application access manager uses the tables, in particular, the configuration table, to identify the database locations and types and send instructions to database drivers.

In step S25 the database drivers copy, move and/or delete data in accordance with the instructions and the adaptor application access manager translates data if the data is copied or moved from one database type to another. When all the data has been moved, copied and/or deleted, in step S26 the adaptor application access manager informs the adaptor application that all instructions have been carried out. In step S27 the adaptor application updates the tables and indicates any new locations of the data records in the network (i.e. updates the data configuration file) and updates the allocation file to specify where each application can be find appropriate data in databases in the distributed databases.

If data has to be moved from one database to another in order to increase performance, instead of moving the data, the data could be copied across one database to the other and subsequently deleted from the original database. This operation ensures that a copy of the data is always available for access. If the data is simply moved, the data may not be available during transit from one database to another. Further, if the movement of data is divided into a two stage copy and delete process, not all of the allocation tables need to be updated simultaneously. Before the deletion of data, some applications can still access data from the old locations until their data allocation tables have been updated to indicate the new locations. Thus dividing the move operation into a copy and delete two stage operation may have advantages in a system which operates a real time adaptive algorithm.

Although the adaption method of Figure 8 has been described as including the moving of data, to optimise the database configuration it is not always necessary to move data between servers. It may be possible to improve performance simply by changing the servers to which queries are sent by one or more of the applications. For example, where data is present for an application on two servers, and the one currently receiving the queries for retrieving data is under a heavy load, the adaption method can simply cause the application query to be

directed to the other server which is less heavily loaded.

Specific adaption algorithms for determining a data distribution and the data servers to which application queries are sent and which can be used to adapt the configuration of the distributed database to improve performance (access time) will now be described.

The methods use tables of figures which represent possible client server connections and related transaction rates and performance figures. These are modelling parameters which represent a model of the distributed database. The performance figures represent costs that are likely to be incurred for a given configuration and are derived from measured values for similar transaction types to the ones being considered.

A model of the distributed database used in the following embodiments includes as parameters forms of both data servers and communications networks so that the performance of the system seen by individual client applications can be estimated for the current load conditions over a range of different access and server configurations. The choice of configuration is determined by an optimisation algorithm which produces solution vectors defining, for each client, which server that client should currently connect to for read access (update access must be made to all servers maintaining a copy of the database).

The algorithm derives costs from retrieval and update rates which can be obtained from the usage files in the embodiments described hereinabove and the degree of contention between retrieval and update transactions, as well as contention between clients on the same server node. In the algorithm it is assumed that the server and communications performance can be based on an MM1 queue. This is based on random transaction arrival rates, exponential service times and the assumption of a single server per client. The principal equations used in the algorithms are:

Response time = 1/((1/BTT)-(1/TIT))........... (1)

where: BTT is the Base Transaction Time, and
TIT is the Transaction Interarrival Time

Overall Transaction Rate (OTR) = ((sum of Retrieval and Update Transaction Rates - Maximum Transaction Rate) x contention value) + Maximum Transaction Rate.... (2)

Thus if the contention value equals 100%, then Overall Transaction Rate equals sum of Transaction Rates.

If the contention value equals zero, the Overall Transaction Rate equals the Maximum Transaction Rate.

For all the contention values, the Overall Transaction Rate will be between the two extremes.

Figure 9 is a schematic illustration of a first database scenario (A) wherein the distributed database comprises 10 databases 100b (A to J) and 10 clients 100a (1 to 10) connected over a network 200 wherein the communication speeds over the network are uniform. Each client 100a resides with a respective database 100b at a node 100 of the network 200. Thus each client can access its respective database without incurring a network delay.

The configuration of the distributed database can be described as a solution vector an example of which is illustrated in Figure 10. The solution vector comprises a string of numbers indexed by client and containing an identification of the server to which each client should access to read data. In this illustrated example it can be seen that a server may be used by more than one client e.g. clients 1 and 8 access server B and clients 3 and 9 access server A.

Two genetic algorithm methods will now be described for determining the optimum solution vector for the scenario illustrated in Figure 9.

Tables 1, 2 and 3 illustrate the modelling parameters used in these two genetic algorithm methods. The algorithm operates for a distributed database as illustrated in Figure 9 i.e. 10 clients and 10 servers and a server contention overlap of 10% is assumed.

TABLE 1

Server Base Transaction Times (BTT) per node (in milliseconds) for database scenario A:

SERVER	BTT
1	>3000
2	>2800
3	>3100
4	>2900
6	>4000
6	>5000
7	>3500
8	>4000
9	>4500
10	>3750

TABLE 2

Client Application rates per node: Retrieval Rate (RR), Update Rate (UR), Overlap % (in events per second) for database scenario A

CLIENT	RR	UR	OVERLAP
1	>0.1	0.01	20
2	>0.2	0.01	20
3	>0.1	0.01	20
4	>0.1	0.01	20
5	>0.15	0.01	20
6	>0.1	0.01	20
7	>0.1	0.01	20
8	>0.15	0.02	2[illegible]
9	>0.1	0.01	15
10	>0.1	0.02	15

TABLE 3

Base Comms Table (time is milliseconds) for database scenario A

Server/ Client	1	2	3	4	5	6	7	8	9	10
1	400	1000	2000	1600	1600	1200	1600	1700	1650	1800
2	1000	400	2000	2000	1800	1900	1750	1650	1800	1900
3	1000	800	400	2000	1750	1600	1670	1760	2000	1900
4	800	1200	2000	400	1850	1900	1950	1850	1900	2000
5	900	1100	1900	1700	400	1800	1600	1760	1640	1860
6	950	1050	1950	1800	1600	400	1550	1750	1750	1800
7	1000	1100	1900	1500	1650	1800	400	1700	1700	1800
8	900	950	1800	1900	1800	1500	1700	400	1900	1950
9	850	1050	1850	1950	1750	1700	1650	1700	400	1900
10	850	1000	1800	1900	1800	1750	1700	1650	1870	400

Figure 11 illustrates a method of a first embodiment of the present invention for modelling the distributed database using a proposed solution vector in order to generate the cost value which, for this embodiment, comprises a worst retrieval rate for the database.

In step S40 the effective transaction rate (ETR) is calculated for each client by combining the retrieval and update rates, using the percentage overlap and equation 2 given hereinabove. In step S41 a 2-dimensional array L is populated with the update rate calculated for that client and indexed by client and server number. Where a server is to be used by a particular client as indicated in the input current proposed solution from the genetic algorithm (step S43), in step S42 the

corresponding array entry in L is replaced with that client's effective transaction rate. A 2-dimensional array C is then created in step S44 to hold effective communication overhead delays. For each client/server interaction rate, now defined in L, the corresponding entry in C is calculated using equation 1 given hereinabove. In step S45, for each server, the client entries in L are used to calculate aggregate loading from all clients of that server using equation 2 given hereinabove. In step S46 this is then converted to an average response time using equation 1 given hereinabove and this is stored in a 1-dimensional array T indexed by server. Any calculated infinite response times are replaced with a suitably large response time. In step S47, for each server, the client with the worst communications overhead delay is found from C and this is added to the relevant entry in T. In step S48 the largest value T is then output and this represents the worst performance of this database configuration.

A first algorithm for generating proposed solution vectors will now be described herein after with reference to Figure 12. This flow diagram illustrates the steps of a Breeder genetic algorithm.

In step S50 an initial random population P is created using a non-binary representation. Each gene position corresponds to a client node and the allele indicates which server that client is to be used for

retrieval access. The maximum number of generations G to be allowed is calculated in step S51 from the following equation:

$$G = 5000 \;/\; ((\text{population size}/2)+1) \ldots\ldots\ldots\ldots \quad (3)$$

In step S52 all the members of the population are then evaluated using the method of Figure 11. In step S53 g is set to 0. In step S54 the current generation number g is incremented by 1 and a loop in the algorithm is entered. All of the numbers of the population are sorted in step S55 based on the evaluation result such that the lowest result is sorted to the top i.e. is the best. The bottom half of the population is then deleted in step S56 and thus the current population p is set to equal half of the total population P. In step S57 the current population p is incremented by 1 and in step S58 two members from the top half of the population are chosen at random and a new number is generated using the technique which will be described hereinafter with reference to Figures 14 and 15. In step S59 using uniformly distributed allele replacement each gene is mutated in the new member based on the defined percentage chance of mutation. In step S60 the new member is evaluated using the procedure of Figure 11 and this is added to the bottom of the population list. In step S61 it is then determined whether the original population

size had been restored i.e. $p = P$ and if not the process returns to step S57. If the original population size P has been restored the process proceeds to step S62 whereupon it is determined whether the maximum number of generations G has been reached i.e. $g = G$. If $g \neq G$ the process returns to step S54. If $g = G$ the process proceeds to step S63 where all the members of the population are sorted based on the evaluation results from the lowest result and best. In step S64 the member of the population with the lowest evaluation result is entered. This can then be used for determining the configuration of the distributed database.

Figure 13 illustrates an alternative method which implements a single three way Tournament genetic algorithm. In step S70 an initial random population is created using a non-binary representation. Each gene position corresponds to a client node and the allele indicates which server that client is to use for retrieval access. In step S71 the maximum number of additional evaluations A to be allowed is calculated using the following equation:

$$A = 5000 - \text{population size} \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (4)$$

In step S72 all the members of the population are evaluated using the method of Figure 11. In step S73 the current evaluation a is set to 0 and in step S74 the

current evaluation A is incremented to enter a loop in the algorithm. Three members of the population are then chosen at random in step S75 and in step S76 these are sorted into BEST, SECOND and WORST. A new member is created in step S77 from BEST and SECOND using the technique described hereinafter with reference to Figures 14 and 15. Each gene in this new member is mutated in step S78 based on the defined percentage chance of mutation using the uniformly distributed allele replacement. A new member is then evaluated in step S79 and in step S80 the new member is used to replace the WORST. In step S81 it is then determined whether the number of additional evaluations has been reached i.e. a = A and if not the process returns to step S74. If the number of additional evaluations has been reached, the process proceeds to step S82 where all the members of the population are sorted based on the evaluation result with the lowest result as BEST. In step S83 the member of the population of the lowest evaluation result is then output. Details on this output member of the population can then be used for the configuration of the distributed database in order to improve the system performance.

Although in both genetic algorithms described above 5000 evaluations are used, any suitable number can be used. Mutation rate and population size can be appropriately selected to tune the genetic algorithm.

For example the mutation rate of 14% can be chosen and the population size of anything from 5 to 500.

The method of generating the new member in Figures 12 and 13 will now be described with reference to Figures 14 and 15.

Using the two parents, in step S90 an initial child is generated as an exact copy of parent 2. A portion of parent 1 of random length and at a random position is then selected in step S91 i.e. length = 5 and position = 8 in this example. This overlay portion is then overlaid onto a portion of the initial child of the same length at another random position in step S92 (i.e. at position = 4 in this example) to generate the resulting child as illustrated in Figure 15.

This technique is a variant of a two-point crossover technique which causes skewing. In this technique allele values in the child are directly overwritten by the overlay portion. There is no splicing and shunting of the genes.

This techniques is not only directly applicable to allelic representations in which the allowed allele range is the same for each gene, where allelic representations are not of the same range, mechanisms can be applied to align the allelic representations.

Figure 16 is a schematic illustration of a second distributed database scenario (B) wherein the nodes 400 and 500 are effectively split into two geographic regions

having low communications costs between nodes in the same region, and high costs between regions. As can be seen in the left hand side in Figure 16 the four nodes 400 comprising data servers 400a (A to D) and clients 400b (1 to 4) are interconnected via a communications network 600 of high speed. Similarly, in the right hand side the six nodes 500 comprising data servers 500a (F to J) and clients 500b (5 to 10) are interconnected via a communications network 700 of high speed. The two high speed communication networks 600 and 700 are interconnected via a low speed communications line 300. Each of the local communications network have a "supernode" whose data performance is 10 times that of the other nodes in the region.

Tables 4 to 6 below illustrate the modelling parameters used in the genetic algorithm methods for this scenario. A server contention overlap of 10% is assumed as for the first scenario (A).

Table 4

Base Comms Table (time in msec)

for Scenario B

Server /client	1	2	3	4	5	6	7	8	9	10
1	400	800	800	800	3000	3000	3000	3000	3000	3000
2	800	400	800	800	3000	3000	3000	3000	3000	3000
3	800	800	400	800	3000	3000	3000	3000	3000	3000
4	800	800	800	400	3000	3000	3000	3000	3000	3000
5	3000	3000	3000	3000	400	1000	1000	1000	1000	1000
6	3000	3000	3000	3000	1000	400	1000	1000	1000	1000
7	3000	3000	3000	3000	1000	1000	400	1000	1000	1000
8	3000	3000	3000	3000	1000	1000	1000	400	1000	1000
9	3000	3000	3000	3000	1000	1000	1000	1000	400	1000
10	3000	3000	3000	3000	1000	1000	1000	1000	1000	400

Table 5

Client application rates per node: Retrieval Rate (RR) Update Rate (UR) Overlap (%) (in events per second) for database Scenario B

Client	RR	UR	Overlap (%)
1	> 0.2	0.005	20
2	> 0.2	0.005	20
3	> 0.05	0.02	5
4	> 0.25	0.005	20
5	> 0.2	0.005	20
6	> 0.1	0.005	20
7	> 0.1	0.005	20
8	> 0.25	0.005	20
9	> 0.05	0.01	5
10	> 0.1	0.005	20

Table 6

Server Base Translation Time (BTT) per Node (in msec) for database scenario B

Server	BTT
1	5000
2	5000
3	500
4	5000
5	5000
6	5000
7	5000
8	5000
9	500
10	5000

Figures 17 to 22 illustrate methods of determining an alternative cost value to the worst performance the cost value as determined in the method described with reference to Figure 11.

The method of Figure 17 is termed "just used" wherein the worst performance is output only for the servers which appear in the solution vector generated by the genetic algorithm.

In step S110 the Effective Transaction Rate (ETR) is calculated for each client by combining the retrieval and update rates, using the percentage overlap and equation 2 given hereinabove. In step S111 a two dimensional array L is populated with the update rate calculated for that client and indexed by client and server number. Where a server is to be used by a particular client as indicated in the input current proposed solution from the genetic algorithm (S113), in

step S112 with corresponding array entry in L is replaced with that clients Effective Transaction Rate. In step S114 the columns in the L array for servers which do not appear in the solution vector are then zeroed to remove their effect. A two dimensional array C is then created in step S115 to hold effective communication overhead delays. For each client/server interaction rate, now defined in L the corresponding entry in C is calculated using equation 1 given hereinabove. In step S116 the columns in the C array for servers which do not appear in the solution vector are then zeroed to remove their effect. In step S117 for each server the client entries in L are used to calculate aggregate loading for all clients on that server using equation 2 given hereinabove. In step S118 this is then converted to an average response time using equation 1 given hereinabove and this is stored in a one dimensional array T indexed by server. Any calculated infinite response times are replaced with a suitably large response time. In step S119, for each server which is represented in the solution vector, the client with the worst communication overhead delays is found from C and this is added to the relevant entry in T. In step S120 the largest value in T is then output and this represents the worst performance of the servers which are to be used by the clients in this database configuration.

Figure 18 illustrates another embodiment termed "plus average" in which 10% of the access times for all nodes is added to the least worst server time. This strikes a balance between minimising the worst performance and aggregate server performance.

In step S130 the Effective Transaction Rate (ETR) is calculated for each client by comparing the retrieval and update rates, using the percentage overlap and equation given hereinabove. In step S131 a two dimensional array L is populated with the update rate calculated for that client and indexed by client and server number. Where a server is to be used by a particular client as indicated in the input current proposed solution from the genetic algorithm (S133), in step 132 the corresponding array entry in L is replaced with that clients Effective Transaction Rate. A two dimensional array C is then created in step S134 to hold effective communication overhead delays. For each client/server interaction rate, now defined in L, the corresponding entry in C is calculated using equation 1 given hereinabove. In step S135, for each server, the client entries L are used to calculate aggregate loading for all clients on that server using equation 2 given hereinabove. In step S136 this is then converted into an average response time using equation 1 given hereinabove and this is stored in a one dimensional array T indexed by server. Any calculated infinite response

times are replaced with a suitably large response time. In step S137, for each server, the client with the worst communications overhead delay is found from C and this is added to the relevant entry in T. In step S138 10% of the average of all the values in T are calculated and in step S139 the largest value in T plus the 10% of the average of all of the average values in T is output as the performance measure for this database configuration.

Figure 19 is a flow diagram of another method of calculating a cost value for the distributed database. This model considers applying updates only to those nodes that are currently being accessed as servers, and adds 10% of the communications access time seen by all clients on the worst server, divided by the number of servers used. This adds a bias based on aggregate user perception but with weighting in favour of over duplication of data. This method can provide enhanced resilience and is referred to as "plus used".

In step S140 the Effective Transaction Rate (ETR) is calculated for each client by combining the retrieval and update rates, using the percentage overlap and equation 2 given hereinabove. In step S141 a two dimensional array L is populated with the update rate calculated for that client and indexed by client and server number. Where a server is to be used by a particular client as indicated in the input current proposed solution from the genetic algorithm (S143), in

step S142 the corresponding array entry in **L** is replaced with that clients Effective Transaction Rate (ETR). In step S144 the columns in the **L** array for the servers which do not appear in the solution vector are then zeroed to remove their effect. In step S145 a two dimensional array **C** is then created to hold effective communication overhead delays. For each client/server interaction rate, now defined in **L** a corresponding entry in **C** is calculated using equation 1 given hereinabove. In step S146 the columns in the **C** array for servers which do not appear in the solution vector is zeroed and in step S147 for each server which appears in the solution vector the client entries in **L** are used to calculate aggregate loading for all clients on that server using equation 2 given hereinabove. In step S148 this is then converted to an average response time using equation 1 given hereinabove and this is stored in a one dimensional array **T** indexed by server. Any calculated infinite response times are replaced with a suitably large response time. In step S152 for each server which is represented in the solution vector the client with the worst communications overhead delay is found from **C** and this is added to the relevant entry in **T**. Also in step S149 the worst server in **T** the average communication times in **C** are calculated and in step S150 10% of the average is taken. This is then divided by the number of different servers appearing in the solution vector in

step S151 and in step S153 the largest value in T plus the value calculated in step S151 is output as the performance value for the database configuration.

Figure 20 is a flow diagram of another method of calculating a cost value for the distributed database. This method is a combination of "just used" and "plus average".

In step S160 the Effective Transaction Rate (ETR) is calculated for each client by combining the retrieval and update rates, using the percentage overlap and equation 2 given hereinabove. In step S161 a two dimensional array L is populated with the update rate calculated for that client and indexed by client and server number. Where a server is to be used by a particular client as indicated in the input current proposed solution from the genetic algorithm (S163), in step S162 the corresponding array entry in L is replaced by that clients Effective Transaction Rate (ETR). The columns in the L array for servers which do not appear in the solution vector are then zeroed in step S164 and in step S165 a two dimensional array C is then created to hold effective communication overhead delays. For each client/server interaction rate now defined in L the corresponding entry in C is calculated in equation 1 given hereinabove. The columns in the C array for servers which do not appear in the solution vector are then zeroed in step S166 and in step S167 for each used

server the client entries in L are used to calculate aggregate loading for all clients on that server using equation 2 given hereinabove. In step S168 this is then converted to an average response time using equation 1 given hereinabove and this is stored in a one dimensional array T indexed by server. Any calculated infinite response times are replaced with a suitably large response time. In step S169, for each server which is represented in the solution vector, the client with the worst overhead communication delay is found from C and this is added to the relevant entry in T. In step S170 10% of the average of all of the values in T are calculated and in step S171 the largest value in T plus the 10% value calculated of the average of all of the values in T is output as the performance value for the database.

Figure 21 is a flow diagram illustrating a further method of calculating a cost value. In this method only used servers i.e. servers which appear in the solution vector the average of all client accesses weighted by their usage rate is added to all used servers. This technique is termed "plus all" and is thought to be the most realistic in terms of representing user perception of quality of service of the distributed database.

In step S180 the Effective Transaction Rate (ETR) is calculated for each client by combining the retrieval and update rates, using the percentage overlap and

equation 2 given hereinabove. In step S181 a two dimensional array L is populated with the update rate calculated for that client and indexed by the client and server number. Where a server is to be used by a particular client as indicated in the input current proposed solution from the genetic algorithm (S183), in step S182 the corresponding array entry in L is replaced by that clients Effective Transaction Rate (ETR). The columns in the L array for servers which do not appear in the solution vector are then zeroed in step S184. A two dimensional array C is then created in step S185 to hold effective communication overhead delays. For each client/server interaction rate, now defined in L the corresponding entry in C is calculated using equation 1 given hereinabove. The columns in the C array for servers which do not appear in the solution vector is then zeroed in step S186. The process then diverges to step S190 in which, for all used servers, the average of the communications times in C is calculated weighted by how often the communication links are used. Also in step S187 for each used server the client entries in L are used to calculate aggregate loading from all clients on that server using equation 2 given hereinabove. In step S188 this is then converted to an average response time using equation 1 given hereinabove and this is stored in a one dimensional array T indexed by server. Any calculated infinite response times are replaced with a

suitably large response time. In step S189 for each server which is represented in the solution vector the client with the worst communications overhead delay is found from C and this is added to the relevant entry in T. In step S191 the largest value in T plus the value calculated in step S190 are output as the performance value for the database.

Figure 22 is a flow diagram of a further method of calculating cost value. This method is similar to the "plus all" technique except that only 10% of the average of all clients accessed weighted by their usage rate is added. This technique is termed "plus 10%".

In step S200 the Effective Transaction Rate (ETR) is calculated for each client by combining the retrieval and update rates, using the percentage overlap and equation 2 given hereinabove. In step S201 a two dimensional array L is populated with the update rate calculated for that client and indexed by client and server number. Where a server is to be used by a particular client as indicated in the input current proposed solution from the genetic algorithm (S203), in step S202 the corresponding array entry in L is replaced with that clients Effective Transaction Rate. In step S204 the columns in the L array for servers which do not appear in the solution vector are zeroed. A two dimensional array C is then created in step S205 to hold effective communication overhead delays. For each

client/server interaction rate, now defined in L, the corresponding entry in C is calculated using equation 1 given hereinabove. In step S206 the columns in the C array for servers which do not appear in the solution vector are then zeroed and the process diverges. In step S210 for all used servers (i.e. the servers which appear in the solution vector) the average of the communications times in C weighted by how often the communication links are used are calculated. Also in step S207 for each server used, the client entries in L are used to calculate aggregate loading from all clients on that server using equation 2 given hereinabove. In step S208 this is then converted to an average response time using equation 1 given hereinabove and this is stored in a one dimensional array T indexed by server. Any calculated infinite response times are replaced by a suitably large response time. In step S209 for each server which is represented in the solution vector the client with the worst communications overhead delay is found from C and this added to the relevant entry in T. In step S211 the largest value in T plus 10% of the value calculated in step S210 is then output as the performance value for this database configuration.

So far the techniques described for obtaining the new solution vectors for use in the evaluation methods of Figures 11 and 17 to 25 have been evolutionary techniques wherein a genetic algorithm is used and thus

the technique of Figures 13 and 15 is used with two parent solution vectors. The present invention is not however limited to evolutionary techniques and is applicable to any search technique which can operate on a solution vector. Two non evolutionary techniques will now be described.

Figure 23 is a flow diagram of a Hill Climbing search technique wherein only one parent solution vector is used. In this non-evolutionary technique there is no "population".

In step S220 a random solution vector is generated initially and this becomes the current solution vector. In step S221 an iteration counter m is set to zero. In step S222 the current solution vector (a randomly generated solution vector) is evaluated using the methods of any one of Figures 11 and 17 to 22 to return a fitness value. The process then enters an iterative loop where in step S223 the iteration counter m is incremented. In step S224 it is determined whether the number of iterations have been complete i.e. $m = M$ where M is the total number of iterations to be carried out. If the number of iterations has been complete in step S225 the current solution vector is output as the optimum solution vector. If $m \neq M$, in step S226 a mutant solution vector is created from the current solution vector as will be described in more detail with reference to Figures 25 and 26. In step S227 the mutant solution vector is evaluated

to return a fitness value. The fitness value for the mutant solution vector is then compared with the fitness value for the current solution vector and if it is better then in step S229 the solution vector for the mutant is set as the current solution vector and the process returns to step S223. If not the current solution vector is kept and the mutant solution vector is discarded and the process returns to step S223.

In accordance with this technique only improvements to the current solution vector are kept. This technique, although simple, suffers from the disadvantage that if there is a localised minimum (or maximum if a maximum is to be found) in the search space, the Hill Climbing technique can determine the optimum solution to lie at the localised minimum rather than at the global minimum.

Figure 24 is a flow diagram of another non-evolutionary search technique termed Simulated Annealing.

In step S230 a random solution vector is generated to become the current solution vector. In step S231 the iteration counter m is set to 0. In step S232 the current solution vector is evaluated using the techniques of any one of Figures 11 and 17 to 22 in order to return a fitness value. In step S233 the process then enters an iteration loop wherein the iteration counter m is incremented. In step S234 it is then determined whether the iteration loop is complete i.e. m = M, where M is the total number of iterations to be carried out. If so, in

step S235 the current solution vector is output as the optimum solution vector. If not in step S236 a mutant solution vector is created from the current solution vector as will be described in more detail hereinafter with references to Figures 25 and 26. In step S237 the mutant solution vector is then evaluated to return a fitness value. In step S238 it is determined whether the fitness value for the mutant solution vector is better than the fitness value for the current solution vector. If it is, in step S239, the solution vector for the mutant is set as the current solution and the process returns to step S233. If it is not, in step S240 the following calculation is made:

$$d = e^{\frac{(f_c - f_m)}{t}} \qquad \text{.........(5)}$$

where f_c is fitness of current best solution,

f_m is fitness of the mutant,

and t is a "temperature" which starts high e.g. 10^6 and decays by a geometric cooling schedule.

In step S241 a random number n between 0 and 1 is then generated and in step S242 it is then determined whether n >d. If not in step S243 the solution vector for the mutant is set as the current solution vector and the process returns to step S233. If so the process returns

to step S233 retaining the current solution vector and discarding the mutant solution vector.

The Simulated Annealing technique has the advantage over the Hill Climbing technique in that a worst solution can be accepted in the search procedure thereby allowing the search process to escape from localised minimum in the search for the global minimum.

The technique for generating the mutant solution vector will now be described with reference to Figures 25 and 26.

In step S301 for the single parent an initial child is generated as an exact copy of the parent. In step S302 from a random start position a portion of random length is selected from the parent as an overlay portion. In Figure 26 it can be seen that the initial start position is 8 and the length of the overlay portion is 5. In step S303 the portion is then overlaid in the initial child on a portion of the initial child of the same length at a random start position to generate an intermediate child. As can be seen in Figure 26 the random overlay position is 4. In step S304 a value in the child is then randomly mutated to generate a mutant solution vector to generate a resulting child. In Figure 26 the first value in the string is changed from A to G.

The use of the technique of the present invention in conjunction with a Tournament genetic algorithm has

been evaluated in comparison with a conventional Hill Climbing technique, a Simulated Annealing technique, a Breeder Genetic Algorithm technique using conventional uniform cross over to generate a new solution vector and a Tournament genetic algorithm also using the uniform cross over technique to generate the new solution vector. Results have been obtained for scenario A as illustrated in Figure 9 and scenario B as illustrated in Figure 16.

Figure 27 shows the results for the five optimisers for scenario A with the basic "least worst" server model. For each of 1000 runs, the fitness of the best solution was noted. The diagram firstly shows the percentage of these 1000 runs that found the known globally optimum solution value (referred to as "On Target" results). The second group of columns shows the percentage of times that the fitness of the best found solution was within 5% of the known globally optimal solution (acceptable in industrial context). The third set of columns shows the percentage of times that the fitness of the best found solution was more than 30% greater (given that lowest fitness value here is best) than the known globally optimal solution (deemed unacceptable in an industrial context). In Figure 27 to 29, the following abbreviations have been used:

HC - Hill Climbing technique

SA - Simulated Annealing technique

BDR - Breeder genetic algorithm technique using uniform cross over

TNT - Tournament genetic algorithm technique using uniform cross-over

SKT - Tournament genetic algorithm using the technique of the present invention for generating a new solution vector.

As can be seen in Figure 27 whilst the tournament genetic algorithm using the technique of the present invention is well behind at 1000 evaluations, it still gives good results at 5000 evaluations.

Figure 28 shows the results for scenario A using the "plus all" model variant (seen as more indicative of user perceived quality of service). Once again, although the tournament genetic algorithm technique employing the technique of the present invention to generate a new solution vector gives unimpressive results at 1000 evaluations, good results were obtained at 5000 evaluations.

Figure 29 shows results for scenario B with the basic, least worst server model. Once again although the tournament genetic algorithm employing the technique of the present invention to generate the new solution vector is unimpressive at one thousand evaluations, the results are good for five thousand evaluations.

Figure 30 shows the results for scenario B with the "plus all" model. This problem clearly gives most

optimisers a large amount of difficulty. At 1000 evaluations, performance is wholly acceptable. However the tournament genetic algorithm technique utilising the technique of the present invention to generate the new solution vector provides goods results 5000 evaluations. Thus as can be seen in Figures 27 to 30, this technique is the only one which is able to give any degree of consistent performance which is critical in an industrial context.

Although the present invention has been described hereinabove with reference to its application to the optimization of the configuration of a distributed database, the technique of the present invention is widely applicable to the optimization of any physical system. The technique for generating a new solution vector exploits a particular feature of the chosen representation namely that good contiguous chunks of allele values in one part of the chromosome may also work well as a building block elsewhere in the chromosome.

The present invention can be applied to the optimisation of a distributed processing system of which a distributed data base is one example. In such a system applications (clients) requiring processing to be carried out at nodes (servers) in a network refer their processing requests to particular nodes. It is this configuration which can be adapted using the present

invention in order to distribute the processing load to improve the system performance. In the system, the operating parameters can comprise transaction or processing rates and network communication times can be used in the model as for the distributed data base embodiment.

The technique has also been applied to a hard benchmark problem in structural chemistry involving finding the two dimensional structure which minimizes the energy (as estimated by Lennard-Jones potentials) of a bonded string of atoms. Here the chromosome represents a list of adjacent bond angles and the true optimum in each case is a close-to spiral structure in which the adjacent angles are similar to each other (but not exactly the same), with repeated contiguous patterns of angles along the string. Coded as a maximisation problem, tables 7 and 8 below indicate the results over ten trials for both ten and twenty atom cases for a tournament genetic algorithm using one point (1 pt) two point (2 pt), uniform (unif), and the inventive cross-over technique (skew).

Table 7

	10 atoms:									
1 pt:	20	19	19	19	19	19	19	19	19	19
2 pt:	19	18	19	18	20	19	18	19	19	19
unif:	19	20	19	17	19	18	18	19	18	20
skew:	20	20	20	20	20	20	20	20	20	20

Table 8

	20 atoms:									
1 pt:	37	40	38	37	41	41	20	42	37	39
2 pt:	41	39	40	39	40	41	40	40	41	42
unif:	43	43	37	39	41	41	39	42	41	42
skew:	47	47	45	47	47	46	46	45	47	47

It can be seen from tables 7 and 8 that the technique of the present invention provides the best solution to this problem both in terms of best value found and repeatability of results indicating the wider applicability of this technique.

A further application of this technique is to the configuration of switched networks wherein information units comprise call routing information for controlling

the switching of network switches to route calls. An example of where usage would change in such a network is where a company decides to offer a special discount on calls in an under utilised part of the communications network at a time later that day. A system could adapt itself to allow efficient access of the appropriate data for the duration of the discount in advance and show that the system is optimised to meet the expected demand. In another example, in a mobile telephone network where a customer moves from one city to another, their data records can follow so that they are close by rather than being accessed slowly across large distances. This can take place by monitoring retrieval performance of the data for the customer and adapting the distribution of data to include the retrieval performance.

Clearly the present invention has a wide range of applications and is particularly suited to optimization techniques with complex cost-values which have a complex search space e.g. more than one minimum or maximum.

The application of the technique in genetic algorithms can be tailored to suit the problem by selecting the mutation rate and population size accordingly.

As has been shown the present invention is particularly suited to optimizing a quality of service metric in a distributed database taking into account both

load on the servers and loads on the links interconnecting the servers and clients.

Although the present invention has been described hereinabove with reference to allelic representations of the solution vector, the present invention is not limited to such representations and is applicable to canonical representations.

The present invention can be implemented as a computer program operating on a standard computer and thus the present invention can be embodied as a storage medium containing processor implementable instructions for controlling a processor to carry out the method. Further, since the computer code can be transmitted in electronic form for example by being downloaded over a network, the present invention can be embodied as an electronic signal carrying computer code for controlling a computer to carry out the method.

Although the present invention has been described hereinabove with reference to specific embodiments, the present invention is not limited to such embodiments and it would be apparent to a skilled person in the art that modifications can be made within the spirit and scope of the appended claims.